The extrapolated Liebmann method for solving partial differential equations is selected for study. With typical computer characteristics in mind, several schemes for organizing the requisite data flow are discussed.

To show the potentialities of timing formulas, as well as their limitations and the problems encountered in their construction, one of the data-flow schemes is treated at length. Kernel programs are included, and timing formulas needed in making comparisons of various configurations of two computers are developed.

Kernel analysis of elliptic partial differential equations

by S. G. Hahn and E. V. Hankam

This paper describes an approach to evaluating computer performance in solving a large class of problems. The procedure leads to the development of a set of formulas which contain problem-dependent and machine-dependent parameters, and which, upon substitution of computer time specifications, yield a measure of speed. These formulas take into consideration those instructions which form the main body of a program and thus are repeated over and over again. We shall refer to this set of instructions, which also includes input/output operations for entering the necessary data into main storage, as the "kernel."

The area we have chosen for this analysis is the numerical solution of partial differential equations, since they cover applications in a great variety of problems arising in physics, chemistry, etc. Within this area, boundary value problems for linear second-order elliptic equations have been selected; these frequently occur in applications and lend themselves to the kind of analysis we propose.

We briefly discuss the numerical solution of partial differential equations and alternative ways of organizing the data flow for a particular problem. Then kernel programs are given for the SYSTEM/360 and the IBM 7094 in order to compare the performances of these computers. General timing formulas are derived that include the time needed for both calculation and read/write operations. Finally, the effect of several input/output devices on the general timing formulas is discussed.

Classification and computational considerations

problem classification

Most frequently arising in partial differential equations are boundary value and initial-value (time-dependent) problems, where mixed initial-boundary-value problems are included with the latter. To solve the differential equations numerically, we first convert them into difference equations and classify them in the following manner:

- Boundary and initial-value problems involving constant coefficients
- Boundary and initial-value problems involving coefficients that depend upon the independent variables
- Initial-value problems involving coefficients that depend both upon the independent and dependent variables

computational considerations

Computational considerations in these three cases mainly differ in input/output operations. In the first case, constant coefficients are kept in main memory, and thus no auxiliary storage is needed.

Difference equations with coefficients that depend upon the independent variables arise either from converting linear differential equations with variable coefficients for any mesh or from using variable mesh size in equations with constant coefficients. These coefficients must be computed at the beginning of the program for every meshpoint, but will not change either at different iterations of a boundary value problem or at different time steps of an initial-value problem. Because of their great number, coefficients usually require too much memory space and, therefore, should be written in auxiliary storage. One or more rows of coefficients are read into main storage at a time for the computation of the corresponding row or rows of unknown functions.

Initial-value problems involving coefficients that depend upon the unknown function are restricted to equations normalized in such a way that the value of the coefficient of the unknown function is always one; the other coefficients are computed for every mesh point at every time step. These coefficients need not be stored, but their calculation may require many computer instructions.

This classification omits the most general case, non-linear equations, because no general theory for the numerical solution of such equations exists at the present time. The literature only contains methods concerning some highly specialized problems.

Higher-order single differential equations, as well as systems of differential equations, can be converted into first-order systems. Assuming that first-order systems can algebraically be solved for one of the partial derivatives (the time derivative in initial-value problems), the numerical calculation is essentially the same as for a single linear equation. Thus, in timing calculations for an electronic computer, the computing time for a single equation is multiplied by the number of unknown functions.

further classification

A further classification of difference equations with respect to the method of solution is:

- Point schemes for boundary value problems
- Explicit schemes for initial-value problems
- Line schemes for boundary value problems
- Implicit schemes for initial-value problems

The main difference between the first two and the second two schemes is that the latter two involve a special kind of matrix inversion. The solution at each point of a boundary value problem could, in principle, also be found by a matrix inversion. However, because of the size of the matrices involved, no suitable methods are available at the present time.

The number of iterations for boundary value problems is a function of ϵ, a chosen small positive number; the iterations proceed until the absolute difference between two consecutive function values is not greater than ϵ for all points in the mesh. The number of time steps for the initial-value problem is determined by the ratio of the time step to the mesh length and by the time span of interest.

boundary value problem considerations

For machine evaluation we choose the numerical solution of the boundary value problem for an elliptic linear partial differential equation of second order with variable coefficients in some domain in the plane. Such problems occur frequently.[1] Thus, we seek the function $u(x, y)$ that satisfies the equation

$$C_1(x, y)u_{xx} + C_2(x, y)u_{yy} + C_3(x, y)u_x + C_4(x, y)u_y + C_5(x, y)u = C_6(x, y)$$

where $C_1(x, y)$ and $C_2(x, y)$ are of the same sign throughout the domain. If we replace all derivatives by centered differences and express the function at any point in terms of its four neighbors, as in the 5-point star representation in Figure 1, we obtain

$$u_{ij} = a_{ij}u_{i-1,j} + b_{ij}u_{i+1,j} + c_{ij}u_{i,j-1} + d_{ij}u_{i,j+1} + f_{ij} \qquad (1)$$

where i indicates the column and j the row of the matrix.

If the mesh length is variable, weight factors appear in the finite differences, and thus are included in the coefficients of the difference equation. The shape of the domain is taken care of by the labeling of the mesh points. Irregular boundaries present a problem; one way to handle them is to obtain by interpolation the values of the unknown function at the grid points closest to the boundary. For the purpose of the kernel analysis, we restrict ourselves to rectangular domains with IJ ($i = 1, \cdots, I$; $j = 1, \cdots, J$) interior mesh points. The subscripts $i = 0, I + 1$ and $j = 0, J + 1$ refer to boundary points.

The coefficients may also indicate the type of boundary value problem we are solving. In a Dirichlet problem, values of the unknown function are prescribed along the boundary of the domain. A Neumann problem has the normal derivative of the

unknown function prescribed on the boundary and the value of the function given at one point. A third boundary value problem has a linear combination of the unknown function and its normal derivative given along the boundary. Both the Dirichlet and Neumann problems are special cases of the third boundary value problem.

In the Dirichlet problem, additional terms appear on the right-hand side of Equation 1 for the mesh points adjacent to the boundary. For the Neumann and third boundary value problems, the difference equations at these points differ from all the others in the domain which results in changing the coefficients for these points.

The coefficients must be initially computed for every mesh point. If their number exceeds the allocated memory space—which is expected to happen in most cases—we write them in auxiliary storage. There are five records of coefficients per row, provided none of them vanishes identically.

In the Richardson method of simultaneous displacements, function values from the previous iterations are exclusively used

$$u_{ij}^{n+1} = a_{ij}u_{i-1,j}^{n} + b_{ij}u_{i+1,j}^{n} + c_{ij}u_{i,j-1}^{n} + d_{ij}u_{i,j+1}^{n} + f_{ij}$$

where the superscript indicates the iteration number. In the Liebmann method of successive displacements, function values already computed replace the old ones

$$u_{ij}^{*n+1} = a_{ij}u_{i-1,j}^{n+1} + b_{ij}u_{i+1,j}^{n} + c_{ij}u_{i,j-1}^{n+1} + d_{ij}u_{i,j+1}^{n} + f_{ij} \qquad (2)$$

The extrapolated Liebmann method of successive overrelaxation determines the new function value as a linear combination of the old and computed values as follows:

$$u_{ij}^{n+1} = \alpha u_{ij}^{*n+1} + (1 - \alpha)u_{ij}^{n} \qquad (3)$$

with $1 < \alpha < 2$ (if $\alpha = 1$, we have the Liebmann method without extrapolation). This method is used later in the paper since it converges when the proper extrapolation parameter is chosen.

In this last scheme, we compute the function at each mesh point from four neighbors using the 5-point star shown in Figure 1. If we want to solve a boundary value problem by the same method, but for a second-order linear elliptic equation

$$C_1u_{xx} + C_7u_{xy} + C_2u_{yy} + C_3u_x + C_4u_y + C_5u = C_6$$

in which the mixed second-order derivative of the unknown function also appears, we would express the function at each mesh point in terms of eight neighbors. We then have the 9-point star shown in Figure 2, and the following equation:

$$u_{ij} = a_{ij}u_{i-1,j} + b_{ij}u_{i+1,j} + c_{ij}u_{i,j-1} + d_{ij}u_{i,j+1} + f_{ij}$$
$$+ g_{ij}u_{i-1,j-1} + h_{ij}u_{i+1,j-1} + k_{ij}u_{i-1,j+1} + l_{ij}u_{i+1,j+1}$$

In this case, there are nine records of coefficients for each row.

Figure 1 5-point star difference scheme

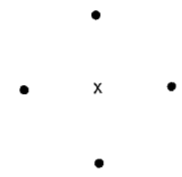


Figure 2 9-point star difference scheme

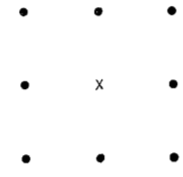
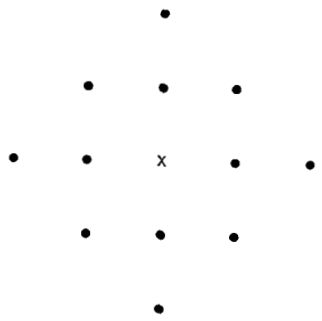


Figure 3 13-point star difference scheme



If the equation is of higher order than two, still more points must be added to the star. For example, the fourth order biharmonic equation

$$u_{xxxx} + 2u_{xxyy} + u_{yyyy} = f$$

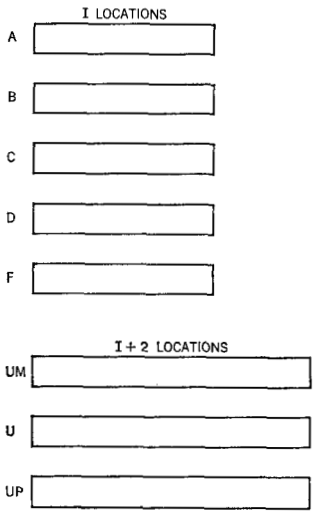
requires a 13-point star as shown in Figure 3 and, consequently, 13 records of coefficients per row.

We described the extrapolated Liebmann method for the boundary value problem in elliptic linear partial differential equations of second order with variable coefficients. The corresponding initial-value problem is handled in the same fashion as the method of simultaneous displacements. Computations for the new time-step in the initial-value problem are equivalent to a new iteration in the boundary value problem. Since the new function values do not immediately replace the old ones in the initial-value case, we have to set aside an extra row for function values if the problem requires auxiliary storage. In addition, all intermediate results should be preserved for output.

In point schemes and explicit schemes, constant coefficients are stored in memory in permanent locations. This also applies to explicit schemes for initial-value problems involving coefficients that depend upon the unknown function. Here, however, we have to compute the coefficients for each mesh point over and over again by using function values from the previous iteration or time step. The remainder of the calculations is the same as in the extrapolated Liebmann method.

Line schemes and implicit schemes require a completely different approach. For boundary value problems, we simultaneously solve for all function values in the same row. Similarly, in an implicit scheme for an initial-value problem, where the (known) function values for the previous time step are expressed in terms of the (unknown) function values for the present time, we are faced with the inversion of a special matrix.

Figure 4 Method 1 data storage allocation



For equations of order not higher than two, the matrix for any row of I elements will be an I by I tridiagonal matrix. It can be factored into two matrices, one consisting of the main diagonal and the next diagonal below, the other consisting of the main diagonal and the next diagonal above; all other elements are zero. The first main diagonal has all ones. Making use of this factorization, two sets of simple linear algebraic equations can be written which are solved recursively.

Line schemes for boundary value problems and implicit schemes for initial-value problems again differ only by the way the coefficients are stored. Otherwise the procedure is exactly the same as in the corresponding point and explicit schemes.

Data flow methods

Assume that a computer has M locations of main storage and that Equation 3 is to be solved in a domain requiring a mesh with $J + 2$ rows and $I + 2$ points per row including boundary points.

Furthermore, we use an s-point star. Let L be the number of memory locations required for instructions necessary to solve the equation, including input/output operations. Then the problem can be solved without auxiliary storage if

$$(I + 2)(J + 2) + sIJ + L < M$$

Generally, $L \ll (I + 2)(J + 2) + sIJ$; the determining factor as to whether auxiliary storage is required is the right-hand side of this inequality.

Let us also assume that the amount of data (unknown functions and coefficients) exceeds the available main storage and thus requires auxiliary storage. Although we use a point-successive computational method, we assume that unknowns and coefficients are read in and written out a row at a time. We further assume that in solving for the unknowns along the jth row, only three rows of data are needed; namely, the $(j - 1)$st, jth and $(j + 1)$st rows.[2]

The most efficient method of arranging the internal flow of data depends upon the characteristics of the computer. To illustrate the problem, we present three alternative methods. Method 1 is efficient on a machine having only one index register, but allowing address modification. Method 2 leads to a program suitable for a computer, such as the 7094, that has at least three index registers. Method 3 can be used on a machine that provides for rapid movement of blocks of data internally.

In Method 1, the program requires $5I$ locations with addresses A, B, C, D, F, for storing the coefficients a_{ij}, b_{ij}, c_{ij}, d_{ij}, and f_{ij}, and $3(I + 1)$ locations with addresses UM, U, UP for storing the unknowns $u_{i,j-1}$, u_{ij}, and $u_{i,j+1}$ as shown in Figure 4. As soon as a row is computed, the addresses of the instructions involving the function values are modified. The row just computed becomes the $(j - 1)$st row for the next calculation. The previous $(j + 1)$st row is computed next, and the new row read in is the new $(j + 1)$st row. Thus the addresses are modified as follows: U becomes UM, UP becomes U, and UM becomes UP. Once this is done, the next row of coefficients is read in over the old ones, and the next row of function values is read into the locations now labeled UP.

In Method 2, $15I$ locations are set aside with base addresses A, B, C, D, F, and $3(I + 2)$ locations with base address U to be used for storing the coefficients a_{ij}, b_{ij}, c_{ij}, d_{ij}, f_{ij}, and the unknowns u_{ij}, as illustrated by Figure 5. New data are read in over data no longer needed, and index registers are used to avoid internal data movement. To illustrate this procedure, let $\mathbf{U}_j$ represent the vector of unknowns in the jth row, i.e.,

$$\mathbf{U}_j = (u_{0j}, u_{1j}, \cdots, u_{I+1,j})$$

and assume that $\mathbf{U}_{j-1}$, $\mathbf{U}_j$, and $\mathbf{U}_{j+1}$ are stored in three consecutive blocks as shown in Figure 6. Index Register 1 is loaded with the base address of $\mathbf{U}_{j-1}$, Index Register 2 with that of $\mathbf{U}_j$, and Index

Figure 5 Method 2 data storage allocation

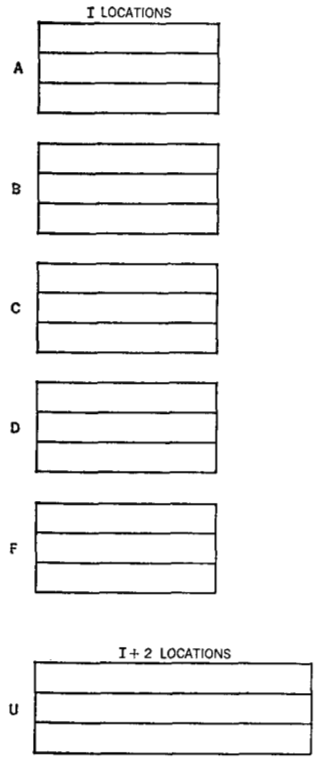


Figure 6 Storage when computing $\mathbf{U}_j$

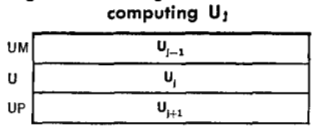
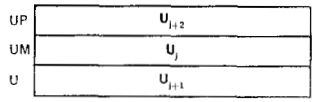


Figure 7 Storage when computing U_{J+1} by method 2



Register 3 with that of $\mathbf{U}_{j+1}$. As points along the jth row are computed, they are written in auxiliary storage. As shown in Figure 7, $\mathbf{U}_{j+2}$ is read in over $\mathbf{U}_{j-1}$, which is no longer needed, and $\mathbf{U}_{j+1}$ is computed. The initial contents of the three index registers are rotated; i.e., Index Register 1 is now used for the base address of $\mathbf{U}_j$, Index Register 2 for that of $\mathbf{U}_{j+1}$, while Index Register 3 must be loaded with the base address of $\mathbf{U}_{j+2}$. The coefficients are treated as the unknowns, except that they need not be written in auxiliary storage.

In Method 3, storage assignment is the same as in Method 1. The three rows of unknowns $\mathbf{U}_{j-1}$, $\mathbf{U}_j$, and $\mathbf{U}_{j+1}$ occupy the same locations. When the computation of a row is completed, $\mathbf{U}_{j-1}$ is written in auxiliary storage, and $\mathbf{U}_j$ and $\mathbf{U}_{j+1}$ are moved into the blocks previously containing $\mathbf{U}_{j-1}$ and $\mathbf{U}_j$, respectively. The new row, $\mathbf{U}_{j+2}$, of data is read into the block previously containing $\mathbf{U}_{j+1}$. These procedures are summarized as follows:

$$\left.\begin{aligned}\mathbf{U}_j &\rightarrow \mathbf{U}_{j-1}\\ \mathbf{U}_{j+1} &\rightarrow \mathbf{U}_j\end{aligned}\right\}\ \text{Internal move}$$

$\mathbf{U}_{j+2} \rightarrow \mathbf{U}_{j+1}$ Read in from auxiliary storage

At the end of each iteration, three rows of unknowns are written in auxiliary storage.

When computing $\mathbf{U}_j$, the storage assignments of the unknowns are as shown in Figure 6; when computing $\mathbf{U}_{j+1}$, the assignments are as shown in Figure 8.

modifications

Unless the computer has small storage capacity or the number of elements in a row is very large, it is normally possible to store more than three rows of the unknown function and one row of coefficients at one time. Then the data flow must be modified.

In general, a computer holds k rows of coefficients and $k + 2$ rows of unknowns if

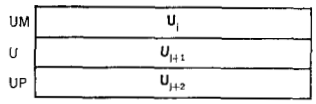
$$(k + 2)(I + 2) + ksI + L < M$$

that is,

$$k < \frac{M - L - 2(I + 2)}{(s + 1)I + 2}$$

One should choose k as a divisor of J.

Figure 8 Storage when computing U_{J+1} by method 3



Methods 1 and 2 are modified so that after the rows 2 to $k + 1$ have been computed and rows 1 to k written in auxiliary storage, rows $k + 1$ and $k + 2$ of the unknown function are moved into rows 1 and 2, and the next k rows are read into rows 3 to $k + 2$ of the blocks. The next k rows of coefficients simply replace the old ones.

With these modifications, Methods 1 and 2 are now essentially identical since a single index register suffices to scan all arrays. This index register is initially set to correspond to the first interior point of the first interior row. It is subsequently stepped by one word until the last interior point of the row is processed.

Before continuing with the calculation of the next row, the index register must be appropriately changed to skip over the boundary points. This procedure is repeated until the entire block is exhausted. In order to make indexing uniform for unknowns and coefficients, it is assumed that a row of coefficients also consists of $I + 2$ elements, although the two (arbitrary) values at the vertical boundaries are actually never used in the calculations. Under the modified procedure, limitations are given by

$$(k + 2)(I + 2) + ks(I + 2) + L < M \tag{4}$$

that is, by

$$k < \frac{M - L - 2(I + 2)}{(s + 1)(I + 2)} \tag{5}$$

In modified Method 3, after the computation of any row is finished, all blocks should be moved up by one row. An additional row should be read into the vacated areas until the last rows have been transmitted to main memory.

additional considerations

Although we have presented three methods in detail, it must not be assumed that these are the only three available. For example, using a computer with four index registers, Method 2 would require the same number of locations as Methods 1 and 3. Other schemes involving, for instance, indirect addressing could be devised. The most efficient method depends upon the characteristics of the computer. However, we feel that the comparison of computers based upon the schemes selected is valid for a wide class of partial differential equations.

Whichever method is used, there is a cost factor associated with setting up the next line of computation. In Method 1 this cost factor is based on address modification, in Method 2 the cost relates to the proper setting of index registers, and in Method 3 the cost is in the internal moving of data.

Kernel programs for SYSTEM/360

short arithmetic

Consider the internal data flow of modified Method 2 for a 5-point star difference scheme. Equations 4 and 5 must be modified for SYSTEM/360, since each memory location holds one byte consisting of 8 bits, and 4 bytes constitute one word.[3] Thus we have

$$4(k + 2)(I + 2) + 20k(I + 2) + L < M \tag{6}$$

and

$$k < \frac{M - L - 8(I + 2)}{24(I + 2)} \tag{7}$$

Addresses A, B, C, D, F refer to the first element in the corresponding block; UM is the address of u_{00} and U that of u_{01}. Whatever the size of the mesh and the memory, six registers are allocated to hold the base addresses of the instruction set and the six arrays. Symbols R0, R1, R2, etc., are used to designate

Table 1 Registers for modified method 2 short arithmetic

*Register**	*Contents*	*Comments*
R15	Addresses of instructions, constants, and a_{01}	
R14	Address of b_{01}	
R13	Address of c_{01}	
R12	Address of d_{01}	
R11	Address of f_{01}	
R10	Address of u_{00}	
R0	4	To increment R2
R6	8	To skip boundary points
R7	$4[(k-1)(I+2)+1]$	To test the end of a block
R9	$4(I+2)$	To test the inner loop
E4	EPSLON	To test convergence
E6	ALPHA	Extrapolation parameter

* R: General register E: Floating-point register

general registers and E0, E2, etc. floating-point registers which need not coincide with the register bearing the same number (e.g., R0 need not correspond with General Register 0).

At the beginning of the program, assume the registers to be loaded as shown in Table 1.

In the kernel program for SYSTEM/360 short arithmetic (Appendix 1), we use the symbols UM for the $(j-1)$st row and U for the jth row; storage allocation for these quantities is illustrated in Figure 4. Instructions executed with the same frequency are grouped together in anticipation of their use in the derivation of timing formulas. Excluded from timing considerations are instructions executed only once per iteration, since a kernel program, by definition, contains only those segments of a program which contribute significantly to the overall timing. Some of these instructions are, however, included in the program to preserve continuity.

long arithmetic

If we consider the internal data flow using long arithmetic and a 5-point star difference scheme, Equations 6 and 7 become, because of the double-word length,

$$8(k+2)(I+2)+40k(I+2)+L<M$$

and

$$k<\frac{M-L-16(I+2)}{48(I+2)}$$

At the beginning of the program, the general registers shown in Table 2 are loaded differently from the corresponding ones in short arithmetic.

Appendix 2 lists those instruction groups for long arithmetic which differ from the ones in Appendix 1 for short arithmetic.

a new last row for each block is read in. Writing should start simultaneously with the computing of the second row. When the whole block is moved up a row, reading into the last row should begin.

Whichever method is used, it is clear that if more than one channel is attached to a computer, the coefficients should be distributed in the best way among input/output devices belonging to the different channels. The two sets of unknowns u_{ij} should be transmitted through two different channels to take advantage of simultaneous reading and writing.

Timing of operations

computation timing

Here we first derive separate timing formulas for computation and input/output operations and then an expression for determining the total time necessary for one iteration assuming overlap between computation and input/output.

Let T_A be the execution time for instructions in Group A (Groups A_1 and A_2) for both the SYSTEM/360 and the 7094; these instructions evaluate u_{ij}^{n+1}, assuming a 5-point star difference scheme.

Let T_A^- be the computing time for any additional point, i.e., T_A^- is the execution time for the following instructions.

- SYSTEM/360 short arithmetic:
  - LE
  - ME
  - AER
- 7094 single precision:
  - LDQ
  - FMP
  - FAD
  - STO

Let T_B be the computing time for testing successive iterations performed by instructions in Group B.

Instructions in Groups A and B constitute the innermost loop. Group A is executed once for each interior point. On the other hand, Group B is not always performed for every interior point of the mesh since $\alpha|u_{ij}^{n+1} - u_{ij}^{n}| \leq \epsilon$ is true for every i, j only in the last iteration. One may, therefore, assume that this test occurs only half the time.

Let T_C be the execution time of instructions required to set up the next row of computation, i.e. Group C. Note that T_C depends upon the method chosen for internal data flow.

Let T_D be the time for initializing the MOVE sequence and, for SYSTEM/360 only, for moving the last words of a block (Groups D_1 and D_2).

Let T_E be the time for moving the last two rows of a block into first positions (Group E).

If we denote the computing time for one interation by T_1

we have

$$T_1 = IJ\{T_A + (s-5)T_{\bar{A}} + \tfrac{1}{2}T_B\} + JT_C + \left(\frac{J}{k} - 1\right)(T_D + pT_E)$$

where

$$p = \begin{cases} 2(I+2) & 7094 \\ \left[\dfrac{I+1}{32}\right] & \text{SYSTEM/360 short arithmetic} \\ \left[\dfrac{I+1}{16}\right] & \text{SYSTEM/360 long arithmetic} \end{cases}$$

(The bracketed symbol represents the "integer" portion of the quantity contained therein.)

In these timing considerations, variable coefficients as well as initial guesses for the solution are assumed to be available in auxiliary storage. There is, of course, a certain amount of computing time involved in calculating the coefficients and writing them, say, on tape. Since this is done only once per problem, the generating time is not included in the overall timing. In addition, any instructions carried out at most once per iteration are also excluded from the timing formulas since their execution time is negligible. The first instruction in Group D_1 (BCT in the SYSTEM/360 and TNX in the 7094 programs) also falls into this category when performed for the last time in any iteration since it is executed J/k times, whereas the remaining instructions in Groups D_1 and D_2 are executed only $J/k - 1$ times as indicated in Equation 9.

In the 7094 programs, the last instruction in Group A_2 (a TXI instruction) is performed J times less frequently than the remainder of the instructions in the same group. To compensate for this, another TXI instruction, the first instruction listed after Group C, has been excluded from the timing of Group C. It should be observed that this TXI, and the preceding three instructions in Group C in the 7094 program (CLA, SUB and STO), are executed only $J - J/k$ times rather than J times as indicated in Equation 9. However, J/k is, in general, a comparatively small number, and this simplification has been made to obtain a uniform timing formula.

I/O time

The computing time T_1 (Equation 9) does not include time required to read information into main memory or to write it into auxiliary storage. If we now define

T_α Total data transmission time
T_β Total access time for positioning a given read/write mechanism
T_2 Total time for input/output operations

then

$$T_2 = T_\alpha + T_\beta \qquad (10)$$

The total access time, T_β, varies according to the input/output device used. The total transmission time, T_α, is the transmission time of one word multiplied by a factor, that depends upon the size of the problem and upon the number of channels available. The total transmission time is calculated by using the following equation:

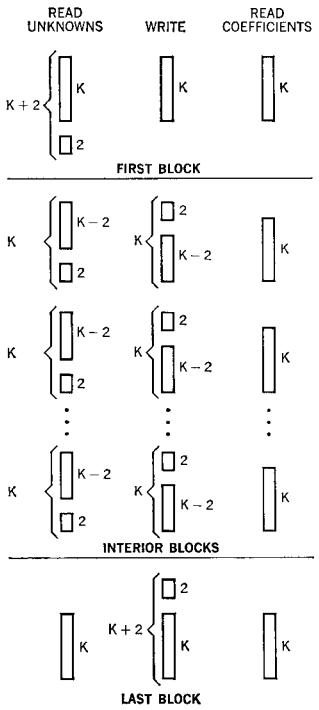
$$T_\alpha = (I + 2)\left\{\left(\left[\frac{S + 1}{l}\right] + 1\right)J + 4\right\}t \tag{11}$$

where

t Time to read or write one word
l Number of channels

Formulas for calculating T_β are given later in this paper.

Figure 9 Layout of data transmission



It should be pointed out that the number of rows of unknowns read in and written out within the same block are not always equal, as shown in Figure 9. The rectangles represent records which consist of the number of rows shown. At the beginning of each iteration, $k + 2$ rows of unknowns are read in and k rows are written out; whereas at the end of each iteration, k rows are read and $k + 2$ rows are written. For interior blocks, the number of rows read and written is k. To allow the proper number of rows to be written in one iteration and read in the next, the records must be of appropriate length. In particular, we write $2J/k - 1$ records per iteration. The first record consists of k rows, the next $2J/k - 3$ records contain alternately 2 and $k - 2$ rows, and the last record contains k rows. Thus, we first read two records of unknowns in order to obtain $k + 2$ rows, and write out one record of k rows. For the next $J/k - 2$ times, we both read and write two records consisting of k rows in each record. At the end of the iteration, the last record with k rows is read in and two records, one containing two rows and the other containing k rows, are written out. Finally, the total time required for computing one iteration in a buffered computer is

$$T = \begin{cases} T_2 & \text{if } T_1 \leq \gamma T_2 \\ T_2 + T_1 - \gamma T_2 = T_1 + (1 - \gamma)T_2 & \text{if } T_1 > \gamma T_2 \end{cases}$$

where γ is the overlap factor, i.e., the percentage of input/output time available for computation.

tape access time

We now describe the use of tapes as auxiliary storage for the coefficients and unknown functions. A minimum number[5] of three data tapes is necessary for the previously described methods: one tape for the coefficients (TC1) and two tapes for the unknowns (TU1 and TU2). Their activity during two consecutive iterations, n and $n + 1$, is given in Table 3. In this case, computation is delayed while the tapes are rewinding

Rewind time can be overlapped with computation by using six tapes: two tapes for the coefficients (TC1 and TC2) and four tapes for the unknowns (TU1, TU2, TU3, and TU4). The six-tape activity is given in Table 4.

Table 3 Activity of three tapes

Iteration	*Operations*
n	Read TC1 Read TU1 Write TU2 Rewind all three tapes
$n + 1$	Read TC1 Read TU2 Write TU1 Rewind all three tapes

Thus far we have assumed that all of the s sets of coefficients are stored on the same tape. This, of course, need not be so, and as many as s different tapes may be available for this purpose.

Using two sets of tapes to eliminate rewind time, and keeping in mind that there are J/k records of coefficients or $2J/k - 1$ records of unknowns on a set of two tapes (see Figure 9), total access time will be

$$T_\beta = \begin{cases} 6a_1 + \left(5\frac{J}{k} - 8\right)a_2 & \text{for} \quad l = 1 \\ 4a_1 + \left(3\frac{J}{k} - 4\right)a_2 & \text{for} \quad 2 \leq l \leq s + 1 \\ 2a_1 + \left(2\frac{J}{k} - 2\right)a_2 & \text{for} \quad l \geq s + 2 \end{cases}$$

where a_1 is the start time from the load point, and a_2 is the record-gap access time.

If the number of tape units available per channel does not allow the doubling of tapes, the timing formulas have to be altered accordingly; i.e., backspace times have to be added, and half of the start times from the load point have to be replaced by record-gap times.

disk access time

Consider now disk storage for input/output operations. Each disk has "tracks," i.e., concentric circles on which information is stored. All tracks of the same radius on a disk storage unit are vertically aligned and form a "cylinder." Information is read or written with the help of a comb-like access arm with a read/write head for each recording surface that can be adjusted to read from or write on any of the cylinders.[6]

Since the access mechanism consists of one read/write head for each track of a cylinder, no mechanical motion of the arm is necessary when reading or writing data records which extend over several tracks on the same cylinder. Therefore, it is advantageous to store data on consecutive tracks of an accessed cylinder rather than on adjacent tracks of a disk surface.

Table 4 Activity of six tapes

Iteration	*Blocks*	*Operations*
n	1 to $\left[\frac{J}{2k}\right]$	Read TC1 Read TU1 Write TU2 Rewind TC1, TU1, TU2
n	$\left[\frac{J}{2k}\right] + 1$ to $\frac{J}{k}$	Read TC2 Read TU3 Write TU4 Rewind TC2, TU3, TU4
$n+1$	1 to $\left[\frac{J}{2k}\right]$	Read TC1 Read TU2 Write TU1 Rewind TC1, TU2, TU1
$n+1$	$\left[\frac{J}{2k}\right] + 1$ to $\frac{J}{k}$	Read TC2 Read TU4 Write TU3 Rewind TC2, TU4, TU3

A single disk unit has, in general, adequate storage capacity for both unknowns and coefficients. The number of cylinders required to store them depends upon the type of disk unit and the size of the problem. To minimize the motion of the access mechanism, storage should be allocated on adjacent cylinders. To keep the timing formulas simple, we assume that a cylinder holds an integral number of blocks of coefficients or unknowns.

Because of the continuous rotation of the disks, the search for the beginning of a record on a track takes an average rotational delay time which must be added to the lateral access time required to locate the addressed cylinder.

Since several arms may be in motion simultaneously, the lateral access time may be reduced if several disk units (or one disk unit equipped with several access mechanisms) are available even though they are attached to the same channel. The availability of several channels permits simultaneous data transmission and, therefore, tends to minimize rotational delay time. When the number of channels is $2 \leq l \leq s + 1$, the data should be distributed in the most economic way. In any case, u_{ij}^{n} and u_{ij}^{n+1} should be transmitted through different channels, while the coefficients should be divided in an optimal manner. In the case of $s + 2$ channels, each coefficient would be transmitted over a separate channel.

Using the notation introduced in Equation 10, we define

$$T_\beta = T_{\beta 1} + T_{\beta 2}$$

where

$T_{\beta 1}$ Total lateral access time (time required for the lateral movement of the arm between cylinders)

$T_{\beta 2}$ Total rotational delay time (time required for the read/write head to reach the desired record on the selected cylinder)

These times may be calculated as follows:

$$T_{\beta 1} = \begin{cases} 3\frac{J}{k}a_1 + (C_u - 1)a_2 & \text{for } l = 1 \text{ and } \delta = 1 \\ 2\frac{J}{k}a_1 + (C_u - 1)a_2 & \begin{cases} \text{for } l = 1 \text{ and } \delta = 2 \\ \text{or } 2 \leq l \leq s+1 \text{ and } \delta = 1 \end{cases} \\ a_1 + (C_u - 1)a_2 & \begin{cases} \text{for } l = 1 \text{ and } \delta \geq 3 \\ \text{or } 2 \leq l \leq s+1 \text{ and } \delta \geq 2 \\ \text{or } l \geq s+2 \text{ and } \delta \geq 1 \end{cases} \end{cases} \quad (12)$$

$$T_{\beta 2} = \begin{cases} \left(3\frac{J}{k} + C_u - 1\right)d & \text{for } l = 1 \\ \left(2\frac{J}{k} + C_u - 1\right)d & \text{for } 2 \leq l \leq s+1 \\ \left(\frac{J}{k} + C_u - 1\right)d & \text{for } l \geq s+2 \end{cases} \quad (13)$$

where

a_1 Time for the access mechanism to move between non-adjacent cylinders

a_2 Time for the access mechanism to move between adjacent cylinders

C_u Number of cylinders to store all unknowns

δ Number of access mechanisms per channel

d Average rotational delay time

If the number of elements in a block exceeds the storage capacity of a cylinder, the timing formulas must be altered accordingly; i.e., for each cylinder-crossing within a block, terms $(J/k)a_2$ and $(J/k)d$ may have to be added to the right-hand sides of Equations 12 and 13, respectively.

For the first iteration (or on restart), the initial movement of the access mechanism may require more time than for subsequent iterations. Since this happens only once per problem, it need not be included in the timing formulas.

drum access time

Drum storage is available as auxiliary storage for SYSTEM/360. A drum with a read/write head for each track is equivalent to one cylinder of a disk unit; there is a rotational delay prior to a read or write operation, but no other access time.[6]

In the following discussion, we assume one drum unit per channel since one drum usually has sufficient capacity to store all data, and no speed is gained by using two or more drums on the same channel. If the problem is so large that more than one drum per channel is needed, each drum should hold an integral number of blocks. Considerations regarding several channels are the same as for disks.

Using the notation of Equations 10, 12, and 13, for drums $T_{\beta 1} = 0$, $C_u = 1$, so that $T_\beta = T_{\beta 2}$ and

$$T_\beta = \begin{cases} 3\frac{J}{k}d & \text{for} \quad l = 1 \\ 2\frac{J}{k}d & \text{for} \quad 2 \leq l \leq s+1 \\ \frac{J}{k}d & \text{for} \quad l \geq s+2 \end{cases}$$

Summary

The procedure described in this paper gives insight into the economic use of computing devices when large blocks of data can be transmitted at the same time. It also demonstrates that computing time can be reduced by an appropriate plan for distributing data among main memory and various input/output devices.

Frequently-used applications in certain scientific areas may emphasize special features of computers that are not particularly important in the procedure we have chosen. For example, in solving partial differential equations by alternating direction methods, data cannot easily be read or written in blocks, and thus the use of a computer with very large main storage even though part of it may have relatively slow access would be preferable to other input/output devices. Although disks offer direct access, seek time would considerably slow down data transmission.

Throughout this paper, it is assumed that only one kind of input/output equipment is available. In reality, this is very rarely the case. For example, one could consider using two types of devices for data transmission and then develop timing formulas for such a configuration. Many such combinations are possible, and the timing equations become more complicated accordingly.

Thus, the widely used and comparatively simple problem chosen for this paper yields rather elaborate timing formulas. For a computer other than the ones discussed, parts of the procedure may have to be modified in order to take advantage of any special feature offered. However, the overall approach essentially remains the same.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Kurt Spielberg, who introduced us to this type of kernel analysis, and to Michael

Held for his contributions during the early stages of the formulation of the problem.

CITED REFERENCES AND FOOTNOTES

1. G. E. Forsythe and W. R. Wasow, *Finite Difference Methods for Partial Differential Equations*, John Wiley and Sons, New York (1960).
2. This is usually true for all second-order equations. In general, for $2k$th order equations, $2k + 1$ rows are required.
3. G. A. Blaauw and F. P. Brooks, Jr., "The structure of SYSTEM/360, Part I—Outline of the logical structure," *IBM Systems Journal* **3**, No. 2, 119-135 (1964).
4. IBM 7094 *Data Processing System*, A22-6703, IBM Data Processing Division, White Plains, New York.
5. In this paper we ignore the possible desirability of using additional tapes or other input/output devices to provide for such unusual conditions as reruns due to errors in writing.
6. IBM SYSTEM/360 *Component Descriptions*, A26-5988-0, IBM Data Processing Division, White Plains, New York.

Appendix 1 Kernel program for SYSTEM/360 short arithmetic

Group		*Instructions*		*Comments*
Initializes R8 once per iteration by loading it with J/k (the number of blocks to be processed in one iteration). Initializes R1 and R2 once per block; sets R1 to $4I$ to test the end of the first row, and sets R2 to 4 for addressing u_{11}.	READ	L	R8, BLOCK	Set R8 to J/k
		.		
		.		
		.		
		Read in u^n_{ij}, coefficients		
		.		
		.		
		.		
		LM	R1, R2, FIRST	Set R1 and R2 to $4I$ and 4
A_1 Computes u^{*n+1}_{ij} from Equation 2 and $\alpha(u^{*n+1}_{ij} - u^n_{ij})$.	INNERL	LE	0, U−4(R2)	Pick up $u^{n+1}_{i-1,j}$
		ME	0, A(R2)	Multiply by a_{ij}
		LE	2, U+4(R2)	Pick up $u^n_{i+1,j}$
		ME	2, B(R2)	Multiply by b_{ij}
		AER	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^n_{i+1,j}$
		LE	2, UM(R2)	Pick up $u^{n+1}_{i,j-1}$
		ME	2, C(R2)	Multiply by c_{ij}
		AER	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^n_{i+1,j} + c_{ij}u^{n+1}_{i,j-1}$
		LE	2, U+4(I+2)(R2)	Pick up $u^n_{i,j+1}$
		ME	2, D(R2)	Multiply by d_{ij}
		AER	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^n_{i+1,j} + c_{ij}u^{n+1}_{i,j-1} + d_{ij}u^n_{i,j+1}$
		AE	0, F(R2)	u^{*n+1}_{ij}
		SE	0, U(R2)	$u^{*n+1}_{ij} - u^n_{ij}$
		MER	0, 6	$\alpha(u^{*n+1}_{ij} - u^n_{ij})$
	NOP1	BC	0, CONT	No operation

Appendix 1 Continued

Group		*Instructions*			*Comments*
B	Compares $\alpha \lvert u_{ij}^{*n+1} - u_{ij}^{n} \rvert$ with ϵ, to test the absolute error.		LPER	2, 0	$\alpha \lvert u_{ij}^{*n+1} - u_{ij}^{n} \rvert$
			CER	2, 4	Compare $\alpha \lvert u_{ij}^{*n+1} - u_{ij}^{n} \rvert$ with ϵ
			BC	12, CONT	If $\alpha \lvert u_{ij}^{*n+1} - u_{ij}^{n} \rvert \leq \epsilon$, skip next instruction
	After an unsuccessful test, the program is changed to skip the entire testing procedure.		OI	NOP1+1, X'FO'	If $\alpha \lvert u_{ij}^{*n+1} - u_{ij}^{n} \rvert > \epsilon$, stop testing
A_2	Computes u_{ij}^{n+1} from Equation 3, and tests to see whether the current row is finished.	CONT	AE	0, U(R2)	$\alpha(u_{ij}^{*n+1} - u_{ij}^{n}) + u_{ij}^{n}$
			STE	0, U(R2)	u_{ij}^{n+1}
			BXLE	R2, R0, INNERL	Set R2 for next element
C	Tests to see if the last row of the block is completed, and increments the row counter.		AR	R1, R9	Set R1 for next row
			BXLE	R2, R6, INNERL	Set R2 for next row
		WRITE		. . .	
			Write out u_{ij}^{n+1}		
				. . .	
D_1	Tests to see if the last block has been processed, and initializes R3, R4, and R5.		BCT	R8, NEXTIT	If last block has been processed, start next iteration
			L	R3, TWOROS	$8(I + 2)$
			L	R4, ADRUM	Address of u_{00}
			L	R5, LASTRO	Address of $u_{0,k+1}$
E	Moves the last two rows of unknowns to the beginning of the block, unless the last block has been processed. This is accomplished by successive transfers of 64 words, the maximum permitted by the MVC instruction. This part of the program assumes $I > 30$.	MOVE	MVC	0(256, R4), 0(R5)	Move 64 words to top
			A	R4, TWO56	Increment addresses for MVC instruction by 256
			A	R5, TWO56	
			S	R3, TWO56	$8(I + 2) - 256$
			C	R3, TWO56	$8(I + 2) - 256 \gtreqless 256$
			BC	2, MOVE	Repeat if more than 64 words remain
D_2	Completes the transfer to the beginning of the block for words in excess of a multiple of 64 and returns to the read sequence.		MVC	0, (REM, R4), 0(R5)	Move remaining words to top
			BC	15, READ	Return to read sequence

Group		*Instructions*		*Comments*
		LM	R1, R2, FIRST	Set R1 and R2 to $8I$ and 8
A_1	INNERL	LD	0, U−8(R2)	Pick up $u^{n+1}_{i-1,j}$
		MD	0, A(R2)	Multiply by a_{ij}
		LD	2, U+8(R2)	Pick up $u^{n}_{i+1,j}$
		MD	2, B(R2)	Multiply by b_{ij}
		ADR	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^{n}_{i+1,j}$
		LD	2, UM(R2)	Pick up $u^{n+1}_{i,j-1}$
		MD	2, C(R2)	Multiply by c_{ij}
		ADR	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^{n}_{i+1,j} + c_{ij}u^{n+1}_{i,j-1}$
		LD	2, U+8(I+2)(R2)	Pick up $u^{n}_{i,j+1}$
		MD	2, D(R2)	Multiply by d_{ij}
		ADR	0, 2	$a_{ij}u^{n+1}_{i-1,j} + b_{ij}u^{n}_{i+1,j} + c_{ij}u^{n+1}_{i,j-1} + d_{ij}u^{n}_{i,j+1}$
		AD	0, F(R2)	u^{*n+1}_{ij}
		SD	0, U(R2)	$u^{*n+1}_{ij} - u^{n}_{ij}$
		MDR	0, 6	$\alpha(u^{*n+1}_{ij} - u^{n}_{ij})$
	NOP1	BC	0, CONT	No operation
B		LPDR	2, 0	$\alpha \lvert u^{*n+1}_{ij} - u^{n}_{ij} \rvert$
		CDR	2, 4	Compare $\alpha \lvert u^{*n+1}_{ij} - u^{n}_{ij} \rvert$ with ϵ
		BC	12, CONT	If $\alpha \lvert u^{*n+1}_{ij} - u^{n}_{ij} \rvert \leq \epsilon$ skip next instruction
A_2	CONT	AD	0, U(R2)	$\alpha(u^{*n+1}_{ij} - u^{n}_{ij}) + u^{n}_{ij}$
		STD	0, U(R2)	u^{n+1}_{ij}
		BXLE	R2, R0, INNERL	Set R2 for next element
D_1		BCT	R8, NEXT IT	If last block has been processed, start next iteration
		L	R3, TWOROS	16 (I+2)
		L	R4, ADRUM	Address of u_{00}
		L	R5, LASTRO	Address of $u_{0,k+1}$
	MOVE	MVC	0(256, R4), 0(R5)	Move 32 words to top
		A	R4, TWO56	Increment addresses for MVC instruction by 256
		A	R5, TWO56	
		S	R3, TWO56	16(I+2) − 256
		C	R3, TWO56	16(I+2) − 256 $\lessgtr$ 256
		BC	2, MOVE	Repeat if more than 32 words remain

Appendix 3 Kernel program for 7094 single precision

Group		*Instructions*		*Comments*
		LXD	BLOCK, 4	J/k
	READ		.	
			.	
			.	
		Read in u_{ij}^{n}, coefficients		
			.	
			.	
			.	
		LXD	FIRST, 2	$k(I+2)-2$
		CLA	ENDFR	$k(I+2)-(I+1)$
		STD	CONT	TXL2, 2, $k(I+2)-(I+1)$
A_1	INNER	LDQ	U−(I+3), 2	Pick up $u_{i-1,j}^{n+1}$
		FMP	A, 2	Multiply by a_{ij}
		FAD	F, 2	Add in f_{ij}
		STO	ERASE	
		LDQ	U−(I+1), 2	Pick up $u_{i+1,j}^{n}$
		FMP	B, 2	Multiply by b_{ij}
		FAD	ERASE	
		STO	ERASE	
		LDQ	U−2(I+2), 2	Pick up $u_{i,j-1}^{n+1}$
		FMP	C, 2	Multiply by c_{ij}
		FAD	ERASE	
		STO	ERASE	
		LDQ	U, 2	Pick up $u_{i,j+1}^{n}$
		FMP	D, 2	Multiply by d_{ij}
		FAD	ERASE	u_{ij}^{*n+1} in AC
		FSB	U−(I+2), 2	$u_{ij}^{*n+1}-u_{ij}^{n}$
		LRS	35	
		FMP	ALPHA	$\alpha(u_{ij}^{*n+1}-u_{ij}^{n})=u_{ij}^{n+1}-u_{ij}^{n}$
		STO	ERASE	
		FAD	U−(I+2), 2	Add u_{ij}^{n}
		STO	U−(I+2), 2	u_{ij}^{n+1}
	TRA	CLA	ERASE	
B		SSP		$\alpha\,\lvert u_{ij}^{*n+1}-u_{ij}^{n}\rvert$
		CAS	EPSLN	
	TRA	CLA		$\alpha\,\lvert u_{ij}^{*n+1}-u_{ij}^{n}\rvert > \epsilon$
	TRA	TXL1		$\alpha\,\lvert u_{ij}^{*n+1}-u_{ij}^{n}\rvert = \epsilon$
A_2	CONT	TXL	TXL2, 2, $k(I+2)-(I+1)$	Test for end of row
		TXI	INNER, 2, −1	Increment for next i
	CLA	CLA	SKIP	TRA CONT
		STO	TRA	Stop testing
		TRA	CONT	
C	TXL2	TXL	WRITE, 2, 1	Test for completion of block
		CLA	CONT	
		SUB	ROW	I+2
		STO	CONT	TXL2, 2, (k−1)(I+2)−(I+1)
		TXI	INNER, 2, −3	Increment for next j

Appendix 3 Continued

Group	*Instructions*		*Comments*
	WRITE	.	
		.	
		.	
		Write out u_{ij}^{n+1}	
		.	
		.	
		.	
D_1	TNX	NEXT, 4, 1	If last block has been processed, start next iteration
	LXD	TWORO, 1	$2(I+2)$
E	MOVE	CLA U+1, 1	u_{0k}
		STO U−$k(I+2)$+1, 1	u_{00}
		TIX MOVE, 1, 1	Increment for next i
D_2		TRA READ	Return to read sequence

Appendix 4 Modifications in kernal program for 7094 double precision

Group	*Instructions*		*Comments*
	LXD	FIRST	$2[k(I+2)-2]$
	CLA	ENDFR	$2[k(I+2)-(I+1)]$
	STD	CONT	TXL2, 2, $2[k(I+2)-(I+1)]$
A_1	INNER DLD	U−$2(I+3)$, 2	Pick up $u_{i-1,j}^{n+1}$
	DFMP	A, 2	Multiply by a_{ij}
	DFAD	F, 2	Add in f_{ij}
	DST	ERASE	
	DLD	U−$2(I+1)$, 2	Pick up $u_{i+1,j}^{n}$
	DFMP	B, 2	Multiply by b_{ij}
	DFAD	ERASE	
	DST	ERASE	
	DLD	U−$4(I+2)$, 2	Pick up $u_{i,j-1}^{n+1}$
	DFMP	C, 2	Multiply by c_{ij}
	DFAD	ERASE	
	DST	ERASE	
	DLD	U, 2	Pick up $u_{i,j+1}^{n}$
	DFMP	D, 2	Multiply by d_{ij}
	DFAD	ERASE	u_{ij}^{*n+1}
	DFSB	U−$2(I+2)$, 2	$u_{ij}^{*n+1} - u_{ij}^{n}$
	DFMP	ALPHA	$\alpha(u_{ij}^{*n+1} - u_{ij}^{n}) = u_{ij}^{n+1} - u_{ij}^{n}$
	DST	ERASE	
	DFAD	U−$2(I+2)$, 2	Add u_{ij}^{n}
	DST	U−$2(I+2)$, 2	u_{ij}^{n+1}
	TRA	DLD ERASE	

Appendix 4 Continued

Group		*Instructions*		*Comments*
A_2	CONT	TXL	TXL, 2, 2[$k(I+2)-(I+1)$]	Test for end of row
		TXI	INNER, 2, -2	Increment for next i
C	TXL2	TXL	WRITE, 2, 2	Test for completion of block
		CLA	CONT	
		SUB	ROW	$2(I+2)$
		STO	CONT	TXL2, 2, 2[$(k-1)(I+2)-(I+1)$]
		TXI	INNER, 2, -6	Increment for next j
D_1		TNX	NEXT, 4, 1	If last block has been processed, start next iteration
		LXD	TWORO, 1	$4(I+2)$
E	MOVE	DLD	U+2, 1	u_{0k}
		DST	U$-2[k(I+2)+1]$, 1	u_{00}
		TIX	MOVE, 1, 2	Increment for next i